COMSC-210 Lecture Topic 0
Course Orientation

Introduction
Program Design & Data Structures
Prereq: Comsc-165 (pointers and linked lists)
Recommmended: Comsc-200 (OOP)
You will learn...
- testing and debugging techniques
- storing, searching, and sorting data
- "smart" arrays and linked-lists
- how to choose the "right" data structure
- how to evaluate algorithm efficiency

About the instructor
- Ph.D. Purdue, Mech.Engr.
- programmer since 1969

class website http://cs.dvc.edu
this is NOT an online class
- attend lecture
- attend lab

Course Goals
Equivalent of UCB CS-61B *but C++*
Prepare for xfer to UC/CSU COMSC program
Understand details of classic *sorting* methods
Understand details of classic *searching* methods
Predict and confirm algorithm efficiency
Be able to arrange data in lists, tables, trees, and graphs
Learn to use "iterators" to search data structures
Become familiar with the C++ STL
Grading policy
- redos, late work, sequence

12 hours per week *(3 lec + 3 lab + 6)*
Learning process: read, quiz, lecture, lab, project
Syllabus and Course Outline (with schedule)
- academic honesty policy

Lecture Period
sign the sign-in sheet (look for "see me")
no electronics during lecture (except photos)
lectures recorded (MP3), files posted
command-line compiling used in lecture
- "lowest common denomiator"
- vendor-, system-, and compiler-independent

Lab Period
Microsoft Visual C++
- on PCs in ATC, ET, and L bldgs (*USB drive recommended*)
- Express 2013 for Web version available for free download
- IDE mode: static library, no precompiled headers

c++ Mac download ; PC MinGW ; PC Cygnus (see instructor)
Upload **.cpp** source files to the COMSC server
- workInProgress, Google Drive, OneDrive, Dropbox

free resources
- DreamSpark accounts
- onedrive.live.com

Textbooks
"C++: Classes and Data Structures" by Jeffrey Childs
- we cover all chapters
- except OOP details in ch.6

other required reading:
- the lecture notes

Console I/O Formatting
formatting numeric output
```
cout.setf(ios::fixed|ios::showpoint);
cout << setprecision(2) ...
```
console and file I/O PDF
`cout.setf(ios::left, ios::adjustfield);` or `left` manipulator
`cout.setf(ios::right, ios::adjustfield);` or `right` manipulator
ref: Burns, Section 4.2

Debugging
"Don't believe everything you think!"
trace and isolate
- using debug output statements w/labels

temporary debug statements: delete or comment out
use // for comments, not /* */ (nesting issue)
try not to spend all your time debugging!

bring "hard" debugging issues to the instructor *in class*
- post files to "workInProgress"

Parameters
"mutable copy" `void fun(Time);` or`int...`
"read-only copy" `void fun(const Time);`
"mutable reference" `void fun(Time&);`
"read-only reference" `void fun(const Time&);`

For-loop Syntax
`for (i; i < 10; i++)` is NOT okay
`i = i++` is NOT okay
Ref: Burns, section 7.3.1

The `NULL` Macro
`NULL` is defined in a #include
use `NULL` for pointer values *only*
or just use `0` (zero) instead

Working With Absolute Values
use the `cstdlib` function
- `abs(x)` for int x

use the `cmath` function
- `fabs(x)` for float x or double x

Storing a DOUBLE in an INT
avoid: `int x = fabs(...);`
prefer: `int x = (int)fabs(...);`

Programming Conventions
mostly C++99, with C++11 auto specifier and range for-loops
C `char`-based arrays used for strings (not C++ `string`s)

use `using namespace std;`
- do *not* use (e.g.) `using std::cout;`

do *not* read `int`s or `double`s directly from console

```
// do NOT use this anymore
int x;
double y;
cin >> x >> y;

// use the "string buffer method" instead
```

www links in the lecture notes
optional: Burns "Programming Concepts In C++"
basic C++ reference

Class Website Tour
Internet URL http://cs.dvc.edu/
student accounts: IDs and passwords
lab assignments and lecture topic notes
online quizzes -- strict time periods
contact instructor via form
some replies to google group

Lab Assignments
Highly structured assignments, 1 per week
Strict about coding practices
debugging techniques
alignment and indenting
about "redos"
no grade until work is complete and correct
no consideration of next lab until current is complete
2 point penalty for not following instructions
5 point penalty for blatant compiler error
10 point late penalty if *any* files missing
no point penalty for "learning opportunities"

just because your program works for you,
it does *not* mean that
you did it right!
programming conventions and lab 0

```
// requires cstdlib and string
int x;
double y;
char buf[100];
cin >> buf; x = atoi(buf); // requires #include <cstdlib>
cin >> buf; y = atof(buf); // requires #include <cstdlib>
```

accommodate "round-off error"
...in calcs using `double`s

```
// do NOT use this anymore (for example)
double y = ...; // result of a calculation
if (y == 100)

// use THIS instead (for example)
if (99.999 < y && y < 100.001)
```

c++ command line compiling with C++11

```
c++ -std=c++11 hello.cpp -Wall
cl hello.cpp /Wall /EHs
```

Ref: Burns, section 2.6

XCode On OSX Yosemite/El Capitan
go to a Terminal session, and type the command: `c++`
If not installed, you'll see a prompt saying so
and inviting you to download and install it
accept the invitation, and in a few minutes...
...you'll have c++ (fully C++11 compatible, too).
Ref: Burns, section 2.2.4

Three program design consideration for COMSC 200

1. Pausing a the end of a program *(optional)*:

```
  cout << "Press ENTER to continue..." << endl;
  cin.get();
} // main -- return 0; not required
```

2. Programmer identification in *all* CPP and H files

```
// Lab LAB NUMBER HERE, LAB TITLE HERE
// Programmer: YOUR NAME HERE
// Editor(s) used: XP Notepad
// Compiler(s) used: VC++ 2010 Express
```

3. Programmer identification in program output

```
int main()
{
  // identifying output statements
  cout << "Lab 1, Problem Solving With C++, Part b\n";
  cout << "Programmer: Joe Student\n";
  cout << "Editor(s) used: XP Notepad\n";
  cout << "Compiler(s) used: VC++ 2010 Express\n";
  cout << "File: " << __FILE__ << endl;
  cout << "Complied: " << __DATE__ << " at " <<  __TIME__ << endl << endl;
  ...
}
```

1. NOTE: Do NOT substitute for __FILE__, __DATE__, or __TIME__. Copy/paste these lines EXACTLY as shown. And yes, those are *double* underscores!
2. NOTE: Code::blocks is NOT a compiler. It's an editor. If you use it, find out what compiler it's using and report that.